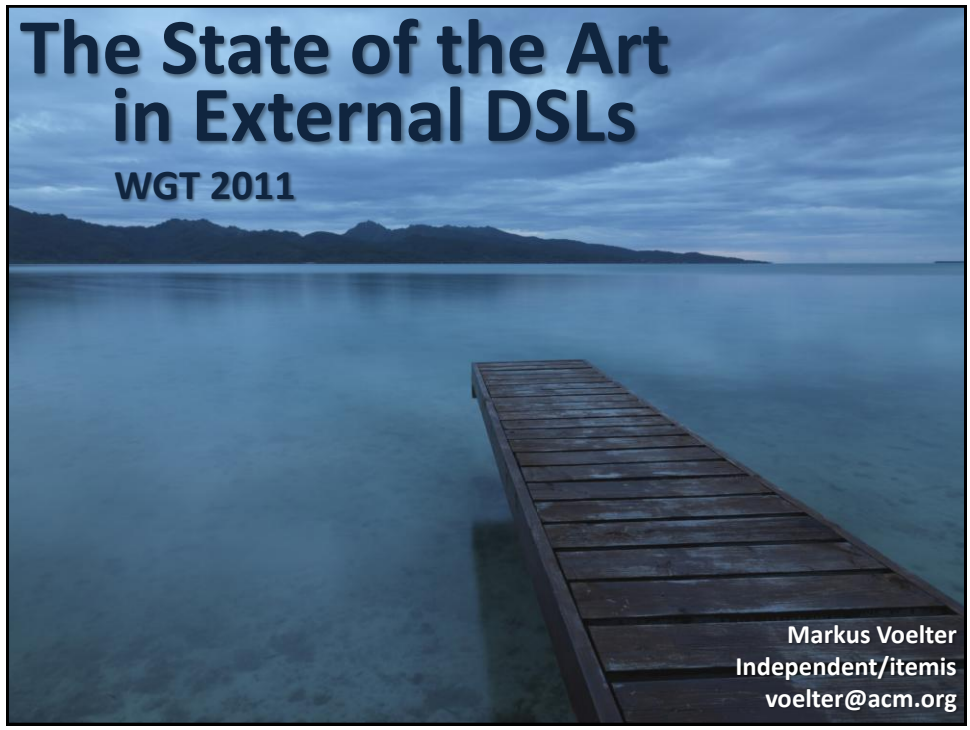
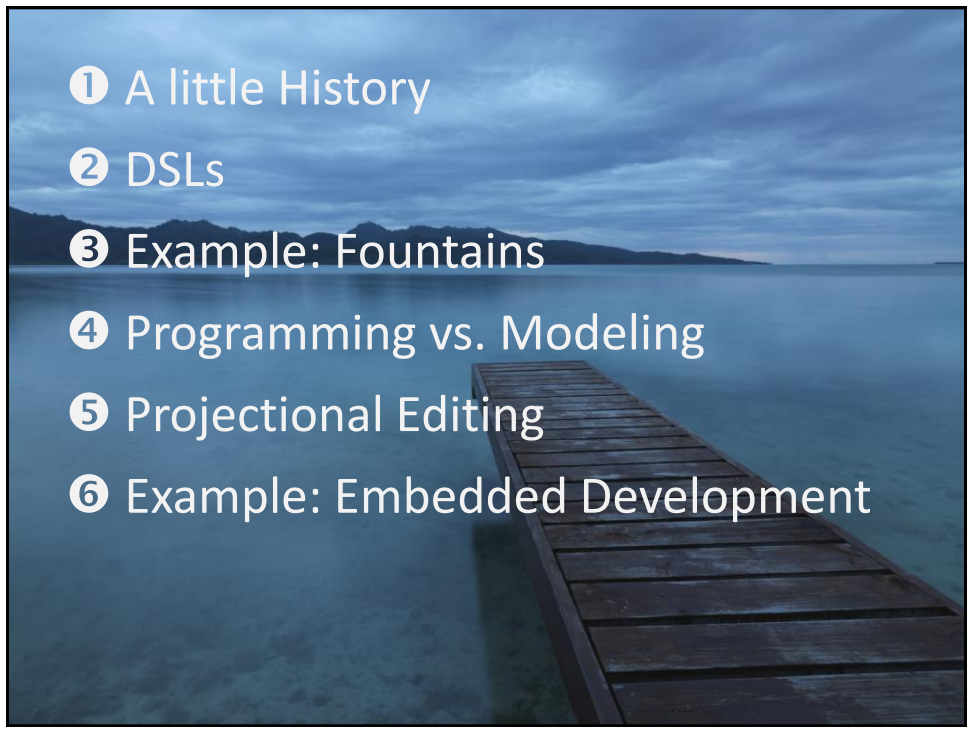
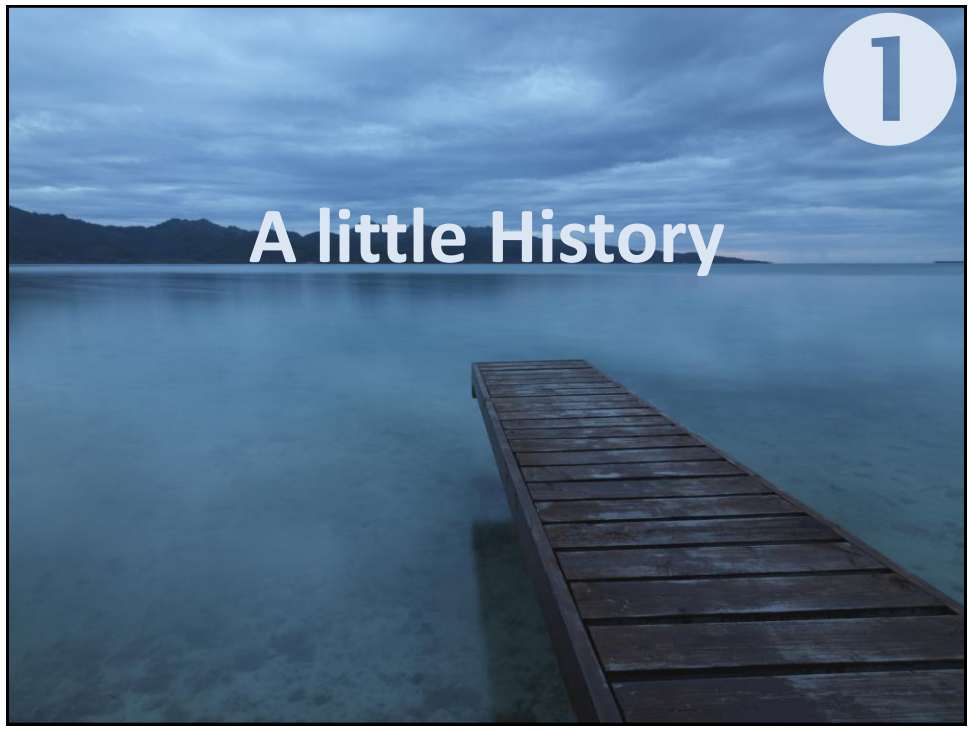
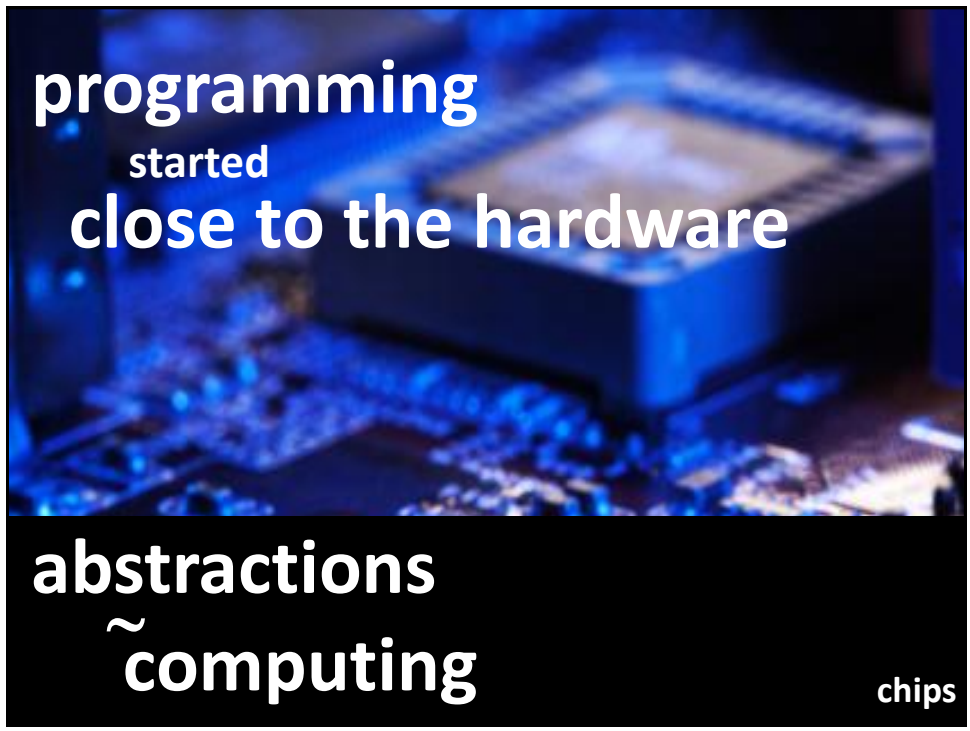

The State of the Art
in External DSLs
WGT 2011
Markus Voelter
Independent/itemis
voelter@acm.org


❶ A little History
❷ DSLs
❸ Example: Fountains
❹ Programming vs. Modeling
❺ Projectional Editing
❻ Example: Embedded Development


1
A little History

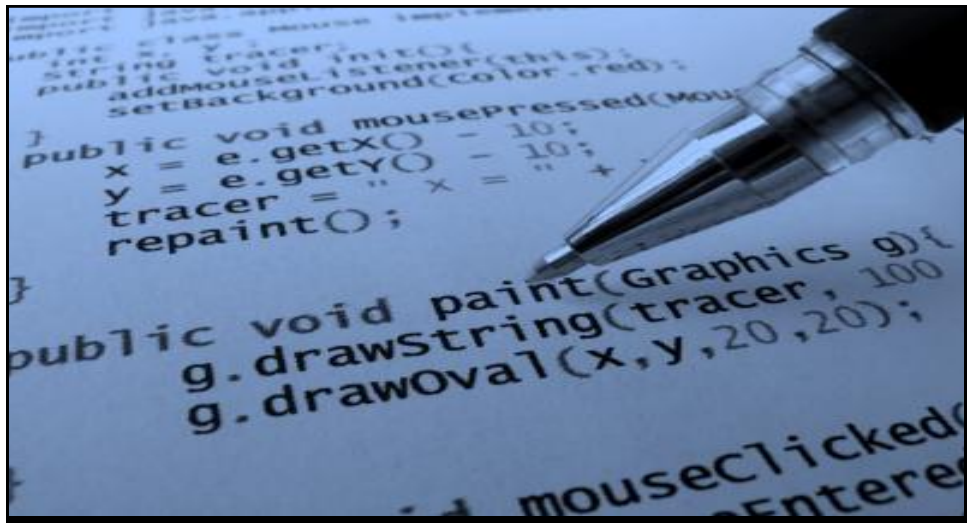

programming
started
close to the hardware
abstractions
~computing
chips

abstractions
~computing

bits

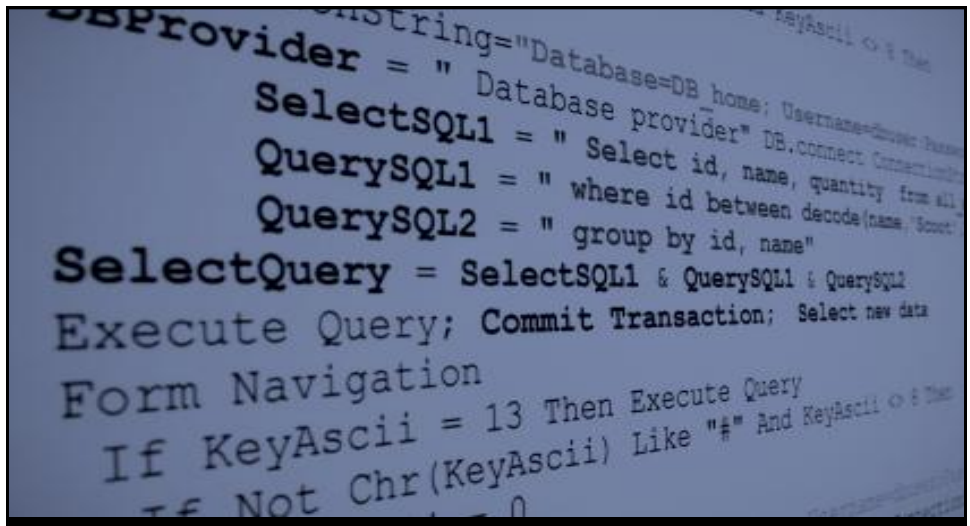
abstractions
~computing

c



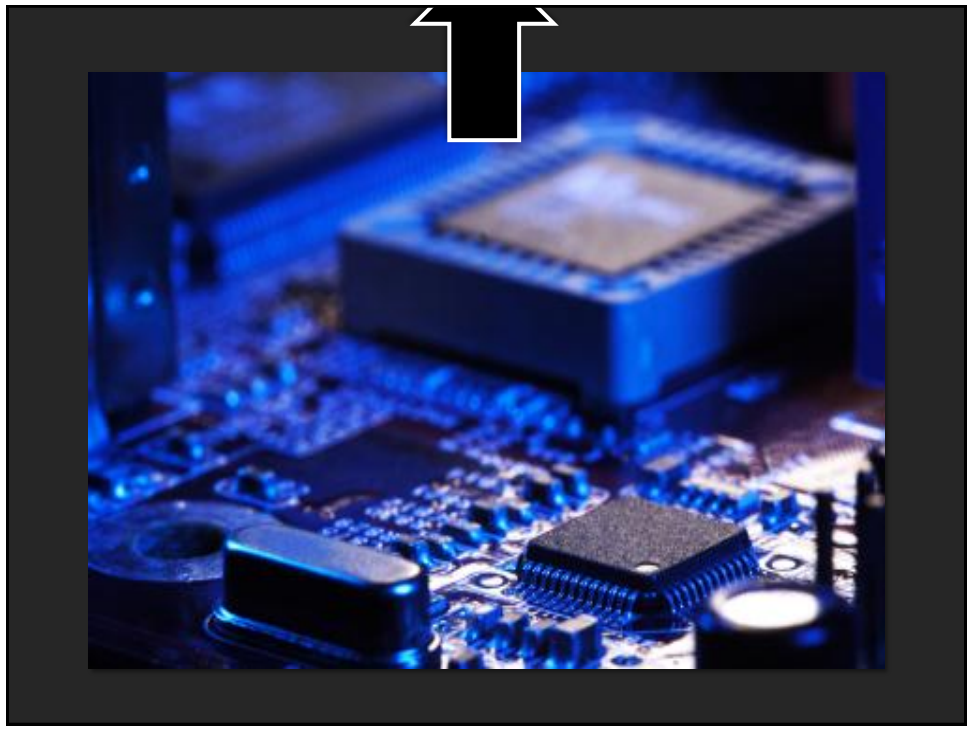
abstractions ~computing?

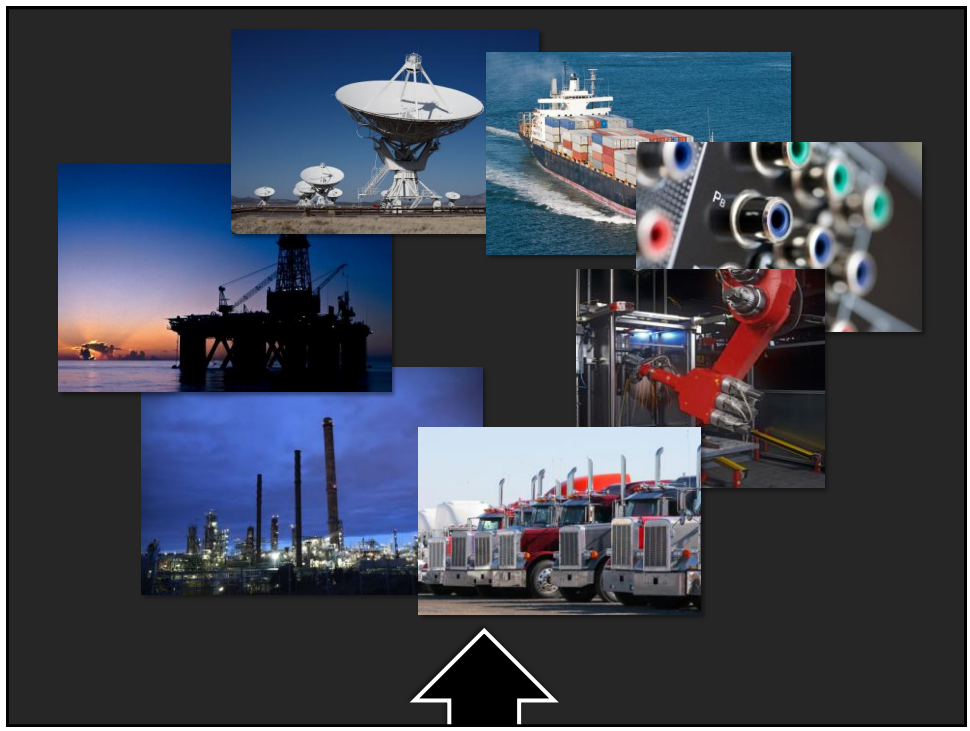
Java



abstractions ~computing?

SQL





general purpose

domain specific

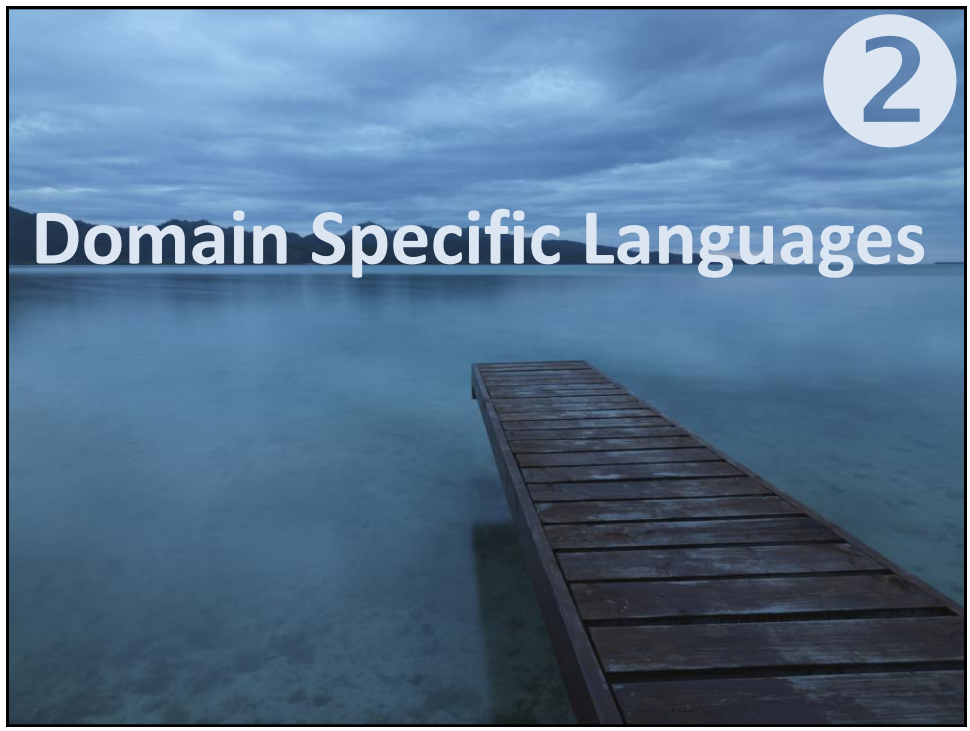
tailor made

effective++

specialized, limited

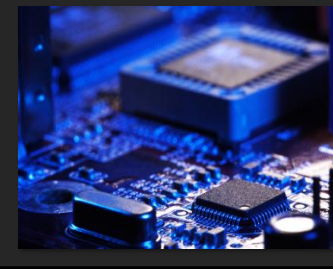
used by experts

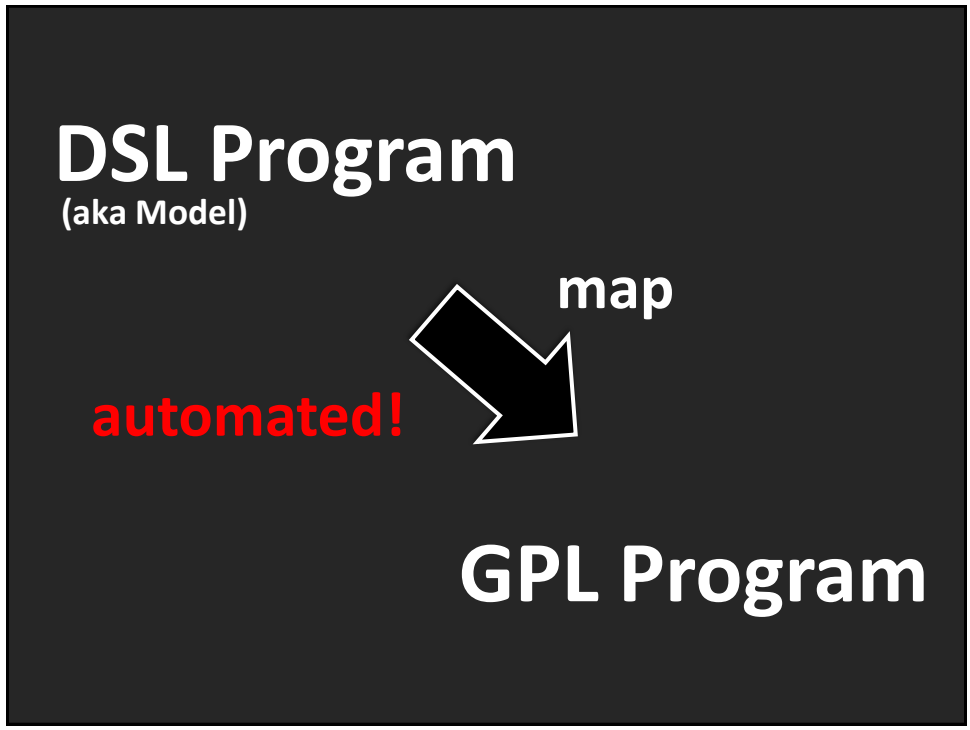
together with other specialized tools

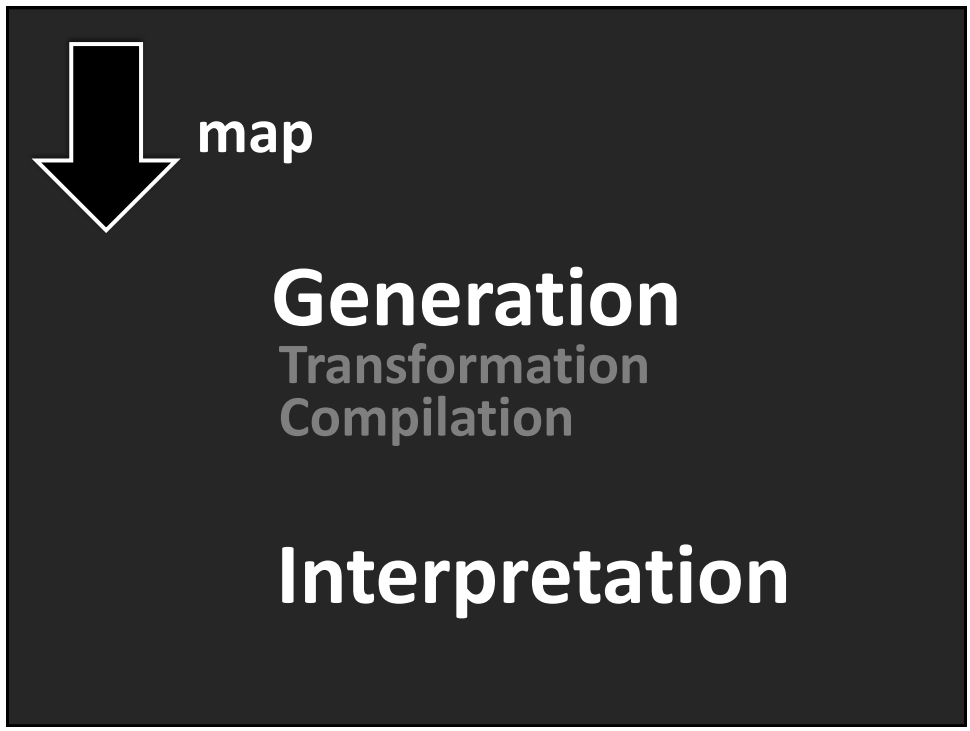


A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

map




DSL Program
(aka Model)
map
automated!
GPL Program


map
Generation
Transformation
Compilation
Interpretation

Activities

- Analysing Domains
- Defining Languages
  - Adapting/Selecting
- Building Editors
- Transforming Models
- Building Generators
- Building Frameworks

Activities

Analysing Domains

Defining Languages

Adapting/Selecting

Building Editors

Transforming Models

Building Generators

Building Frameworks

… and using all of that to build apps

CONTEXT

SOLUTION

Hardware Structure

```
feature BasicOnePump
    pump compartment cc1
        static compressor c1

feature AtLeastOneZone extends BasicOnePump
    water compartment comp1
        pumped by c1
        compartment levelsensor ct_f1
        light l_f1

feature[f] SuperPowerCompartment
    water compartment adds to f
        superPowerMode

feature WithAlarm
    level alarm a1

fountain StdFountain extends AtLeastOneZone
```

SOLUTION

Behaviour

```
pumping program P1 for AtLeastOneZone + WithAlarm +
                       SuperPowerCompartment[f=comp1]  {
    parameter defaultWaterLevel : int
    parameter superWaterLevel: int
    event superPowerTimeout

    init {
        set comp1->targetHeight = defaultWaterLevel
    }

    start:
        on (comp1->needsPower == true) && !(comp1->isPumping) {
            do comp1->pumpOn
        }
        on comp1->enough {
            do comp1->pumpOff
        }
        on comp1.superPumping->turnedOn {
            set comp1->targetHeight = superWaterLevel
            raise event superPowerTimeout after 20
        }
        on comp1.superPumping->turnedOff or superPowerTimeout {
            set comp1->targetHeight = defaultWaterLevel
        }
}
```

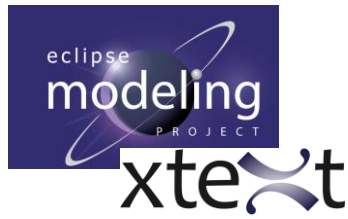
SOLUTION

Plus:

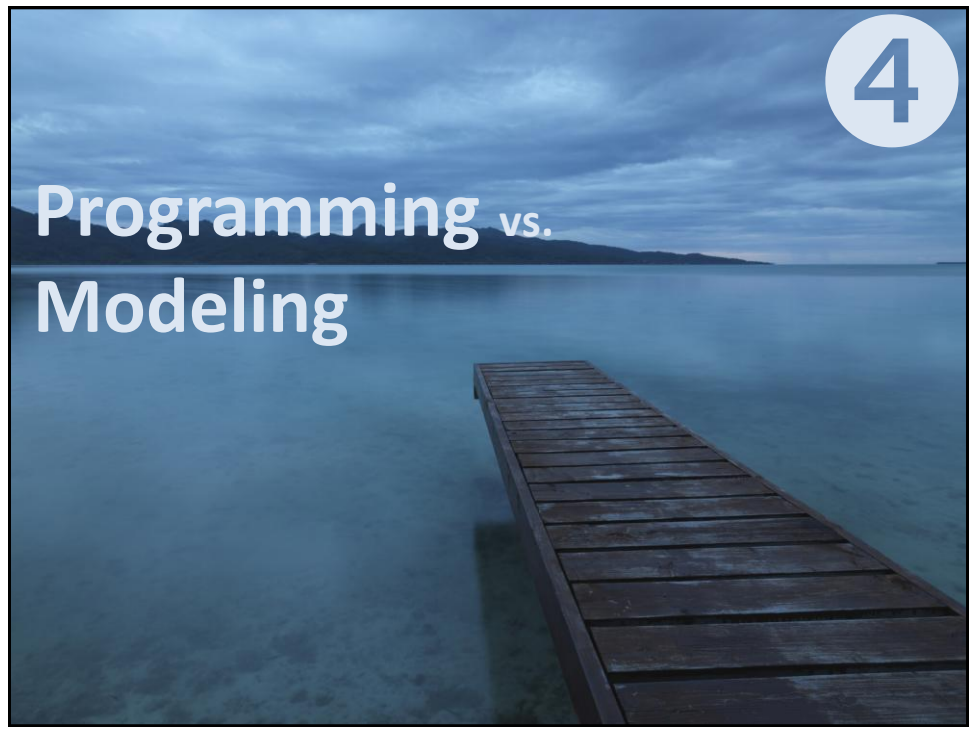
In-IDE Simulator

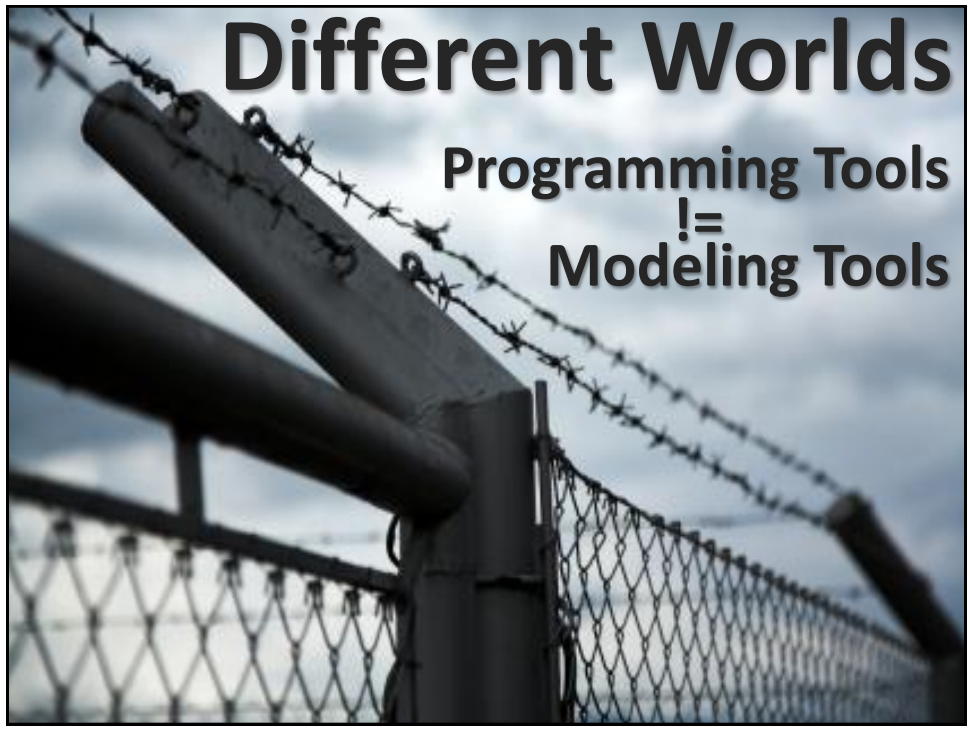
Unit Test Support

TOOLS



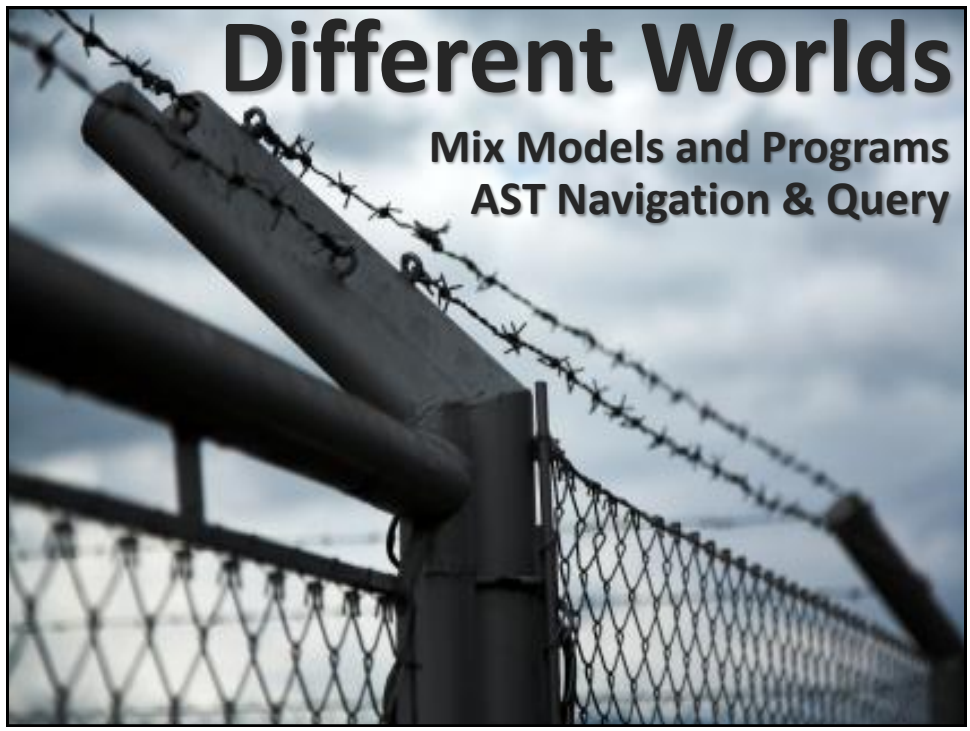
Eclipse Modeling
Eclipse Xtext

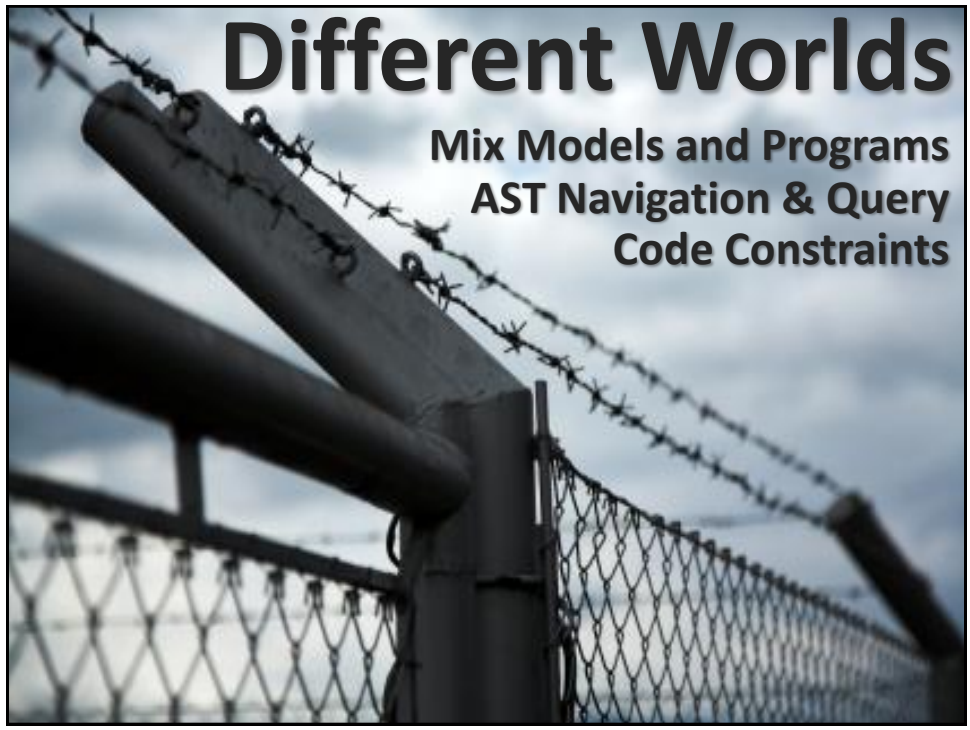



Different Worlds
Programming Tools
!=
Modeling Tools

Different Worlds
Modeling Tool
!=
Modeling Tool

Different Worlds
Mix Models and Programs


Different Worlds
Mix Models and Programs
AST Navigation & Query



Why
the difference?

Modeling

Programming

… (Mostly) Textual Notations

… Concrete Syntax Storage

… (Fancy) ASCII Editors

… Read-Only Visualizations

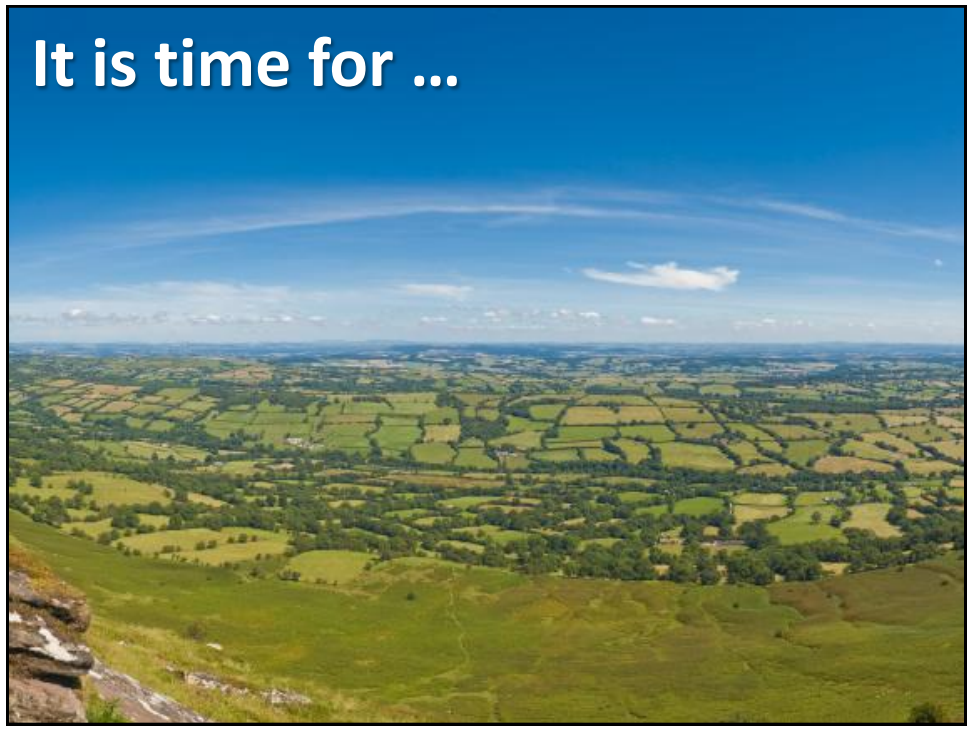
Modeling

- … (Mostly) Graphical Notations
- … Abstract Syntax Storage
- … Projecting Editors
- … Different editable views for model

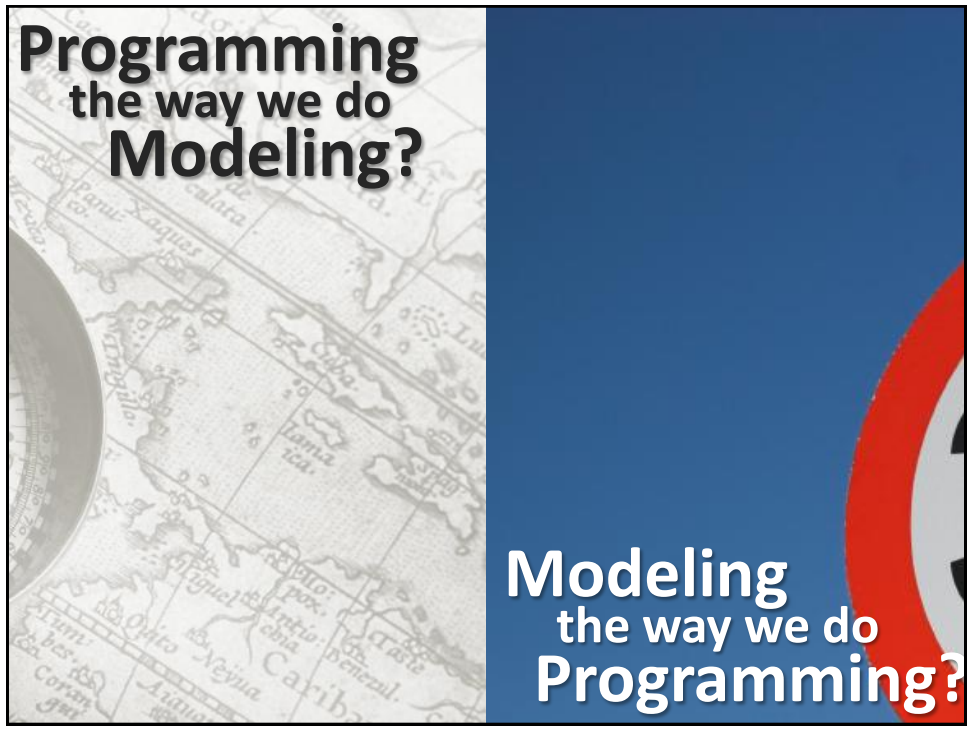
Programming

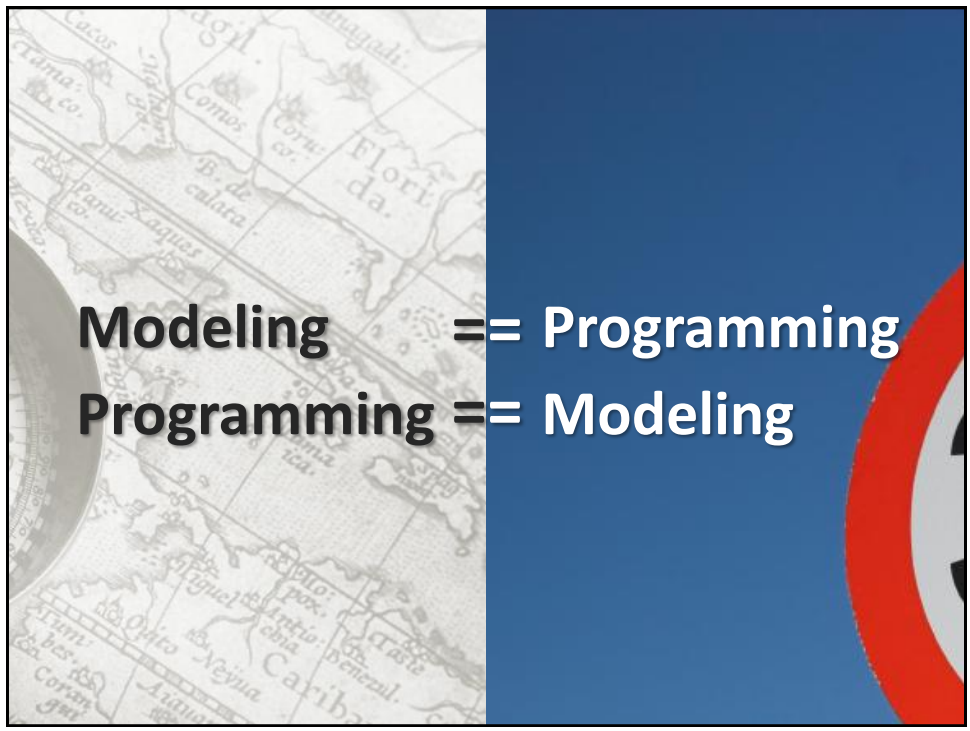
- … (Mostly) Textual Notations
- … Concrete Syntax Storage
- … (Fancy) ASCII Editors
- … Read-Only Visualizations

Why
the difference?


It is time for …

… a Different Perspective


Programming
the way we do
Modeling?
Modeling
the way we do
Programming?


Modeling == Programming
Programming == Modeling

We don‘t want to
model,
we want to
program!

We don‘t want to
model,
we want to
program!

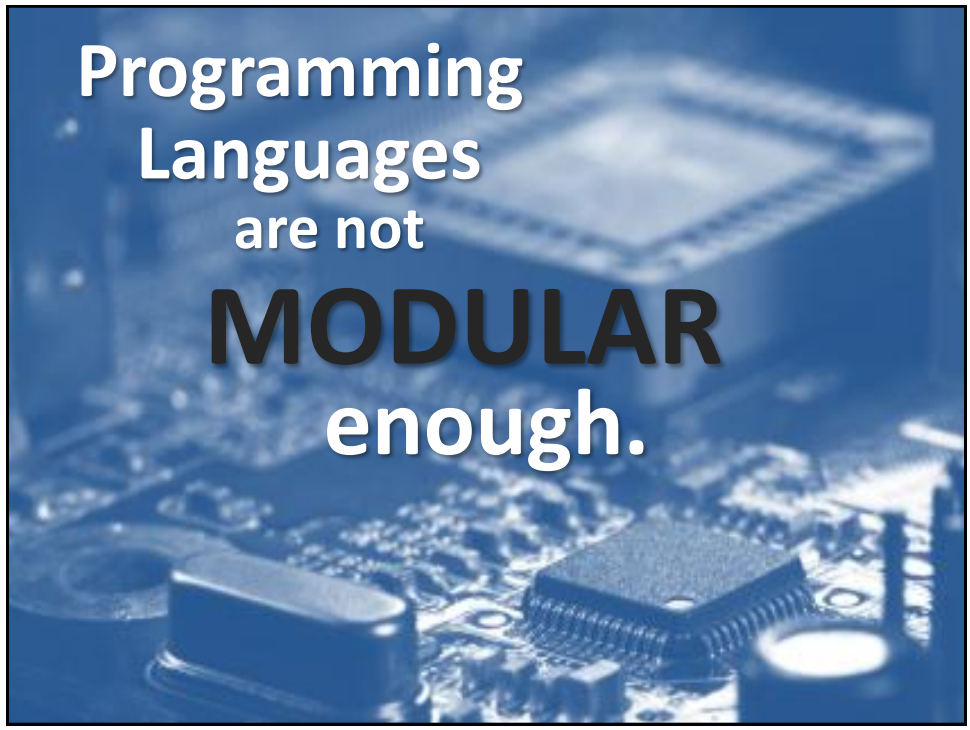
… at different levels of **abstaction**
… from different **viewpoints**
… **integrated**!

We don‘t want to
model,
we want to
program!

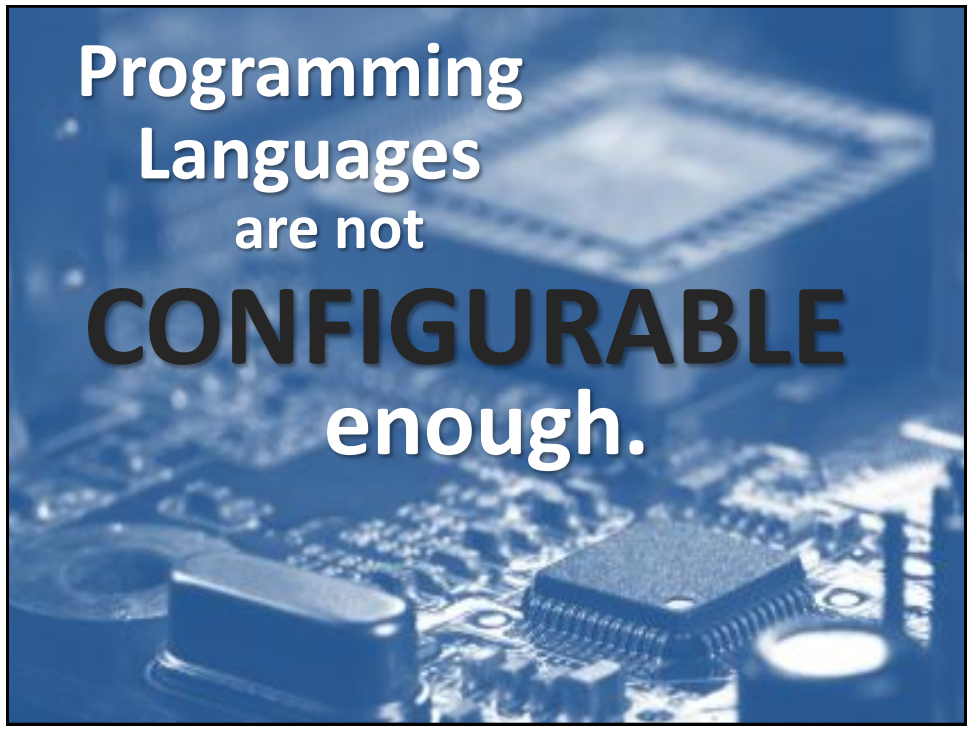
… with different degrees of
domain-specificity
… with suitable notations
… with suitable expressiveness

We don‘t want to
model,
we want to
program!

And always:
precise and tool processable


Programming
Languages
are not
MODULAR
enough.

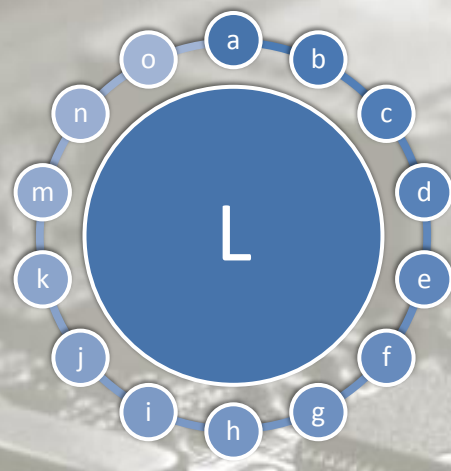
Programming
Languages
are not
COMPOSABLE
enough.


Programming
Languages
are not
CONFIGURABLE
enough.

Programming
Languages
are not
ADAPTABLE
enough.

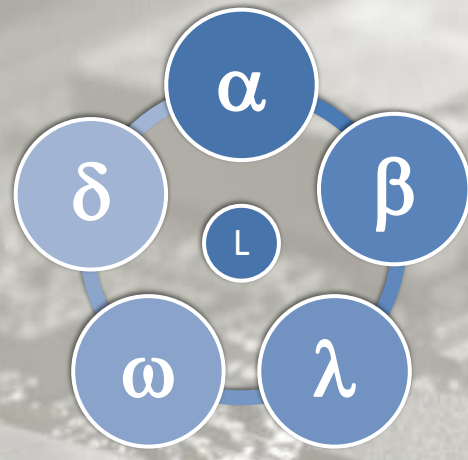
Programming
Language Syntax
is not
FLEXIBLE
enough.

Big Language?



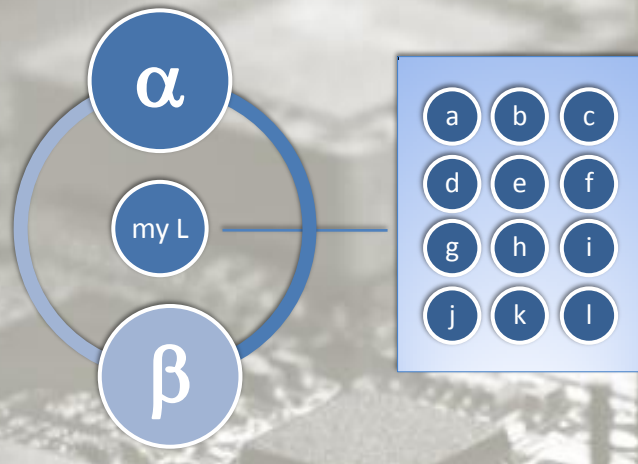
with **many** first class concepts!

Small Language?



with a few, orthogonal and poweful concepts

Modular Language

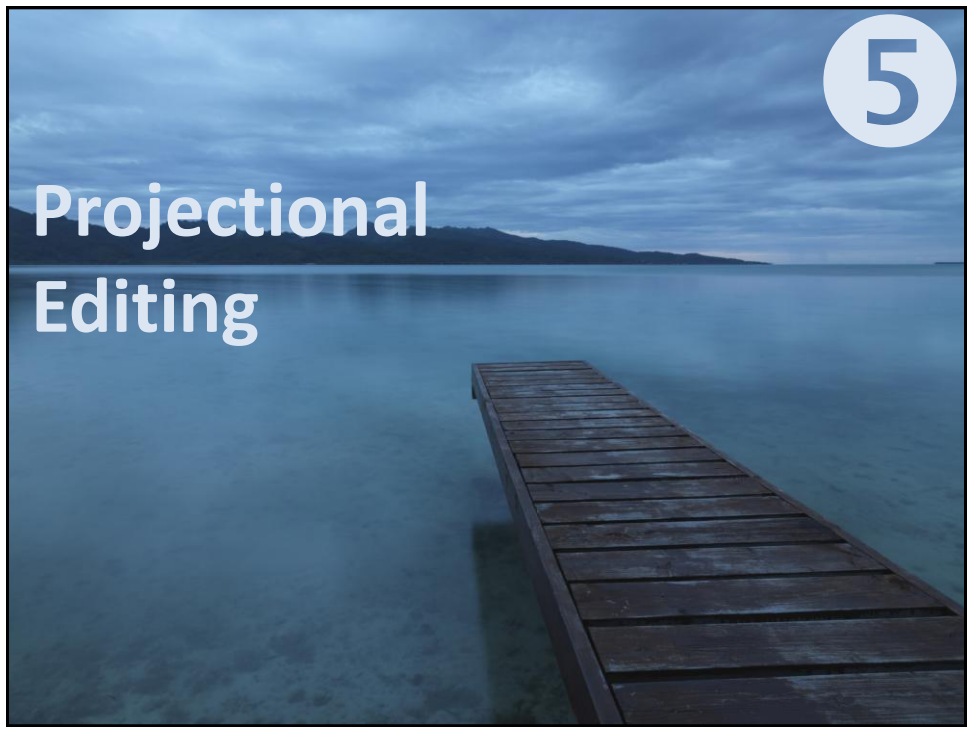


with many optional, composable concepts

Modular Language

Like frameworks
and libraries,

but with syntax
and IDE support

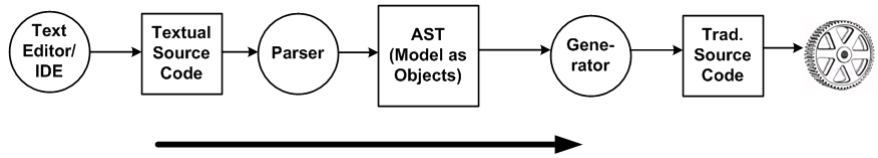


Parser-based

text

… to tree

… to text



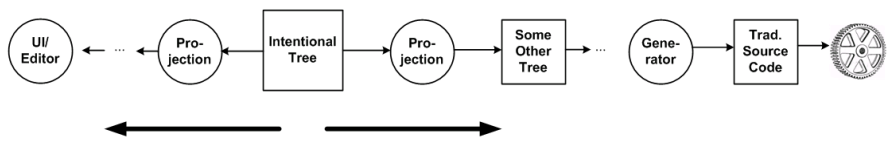
Projectional

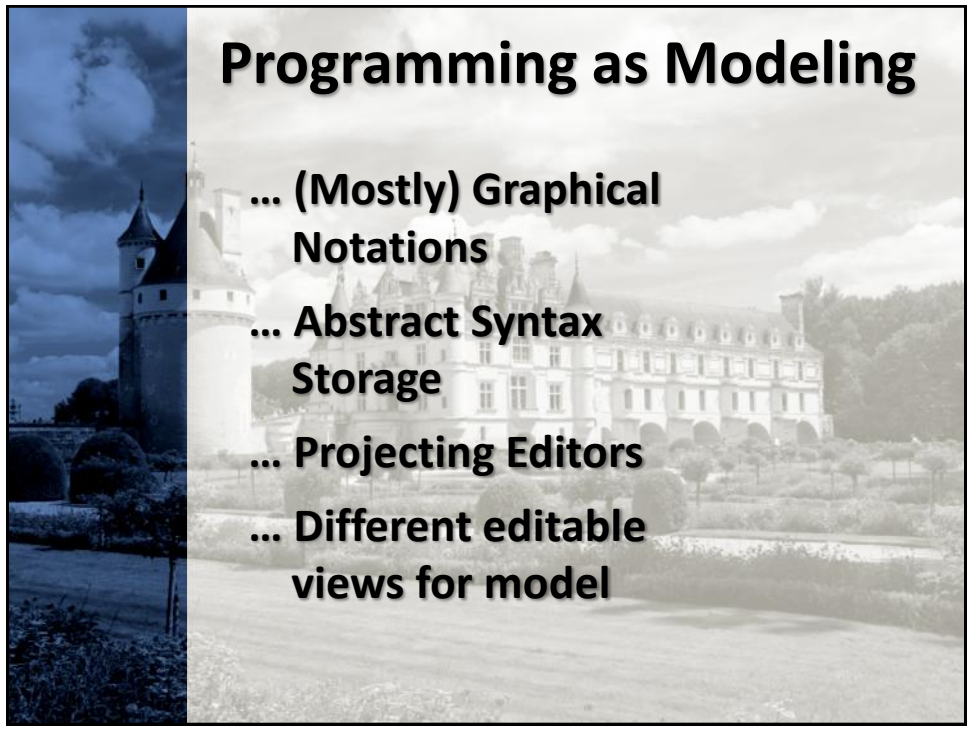
tree

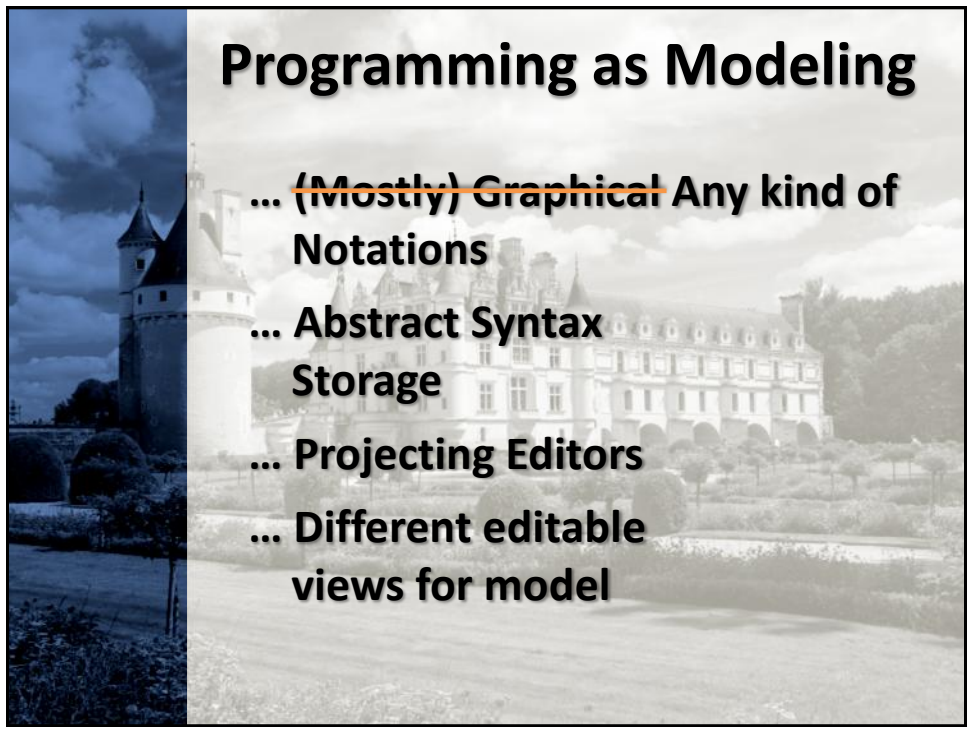
… to text-lookalike (editor)

… to other trees … [*]

… to text




Programming as Modeling
… (Mostly) Graphical Notations
… Abstract Syntax Storage
… Projecting Editors
… Different editable views for model


Programming as Modeling
… (Mostly) Graphical Any kind of Notations
… Abstract Syntax Storage
… Projecting Editors
… Different editable views for model

Language Composition

There‘s no parsing.

Unique Language Element Identity.

Unlimted language composition.

Flexible Notations

Textual
like ASCII

Graphical
box & line

Semi-Graphical
mathematical

} treated the same
can be mixed

Automatic IDE Extension

tool support is inherent
for languages build with
projectional tools

language definition
implies
IDE definition

Multiple Notations

… for the same concepts

e.g. in different contexts
or for different tasks

Partial
Projections

… different views
… for different roles/people
… only a particular variant

Live
Programs

think: spreadsheet

a change to one part of program can lead to (dependent) changes in other parts

Tree Editing

… is different from editing text

… try to make it feel like text

… takes some getting used to

but: for more flexible notations
a more general editing paradigm
is needed

Infrastructure Integration

… storage is not text

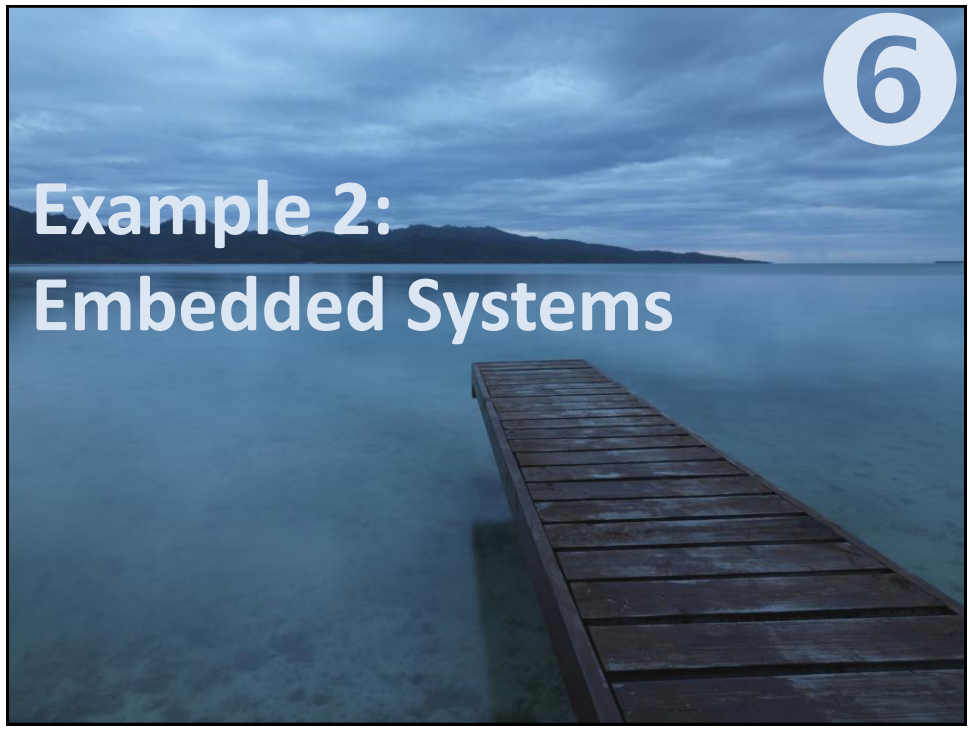
… diff/merge must be in tool

… existing text tools don‘t work

Proprietary Tools

… no standards

… no interop



Example 2:
Embedded Systems

EXAMPLE CASE

SOLUTION

```
doc This module represents the code for the line follower lego robot. It has a coupl
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  constant int WHITE = 500;

  constant int BLACK = 700;

  constant int SLOW = 20;

  constant int FAST = 40;

  doc Statemachine to manage the t
  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> running
    }
    state running {

    }
  }

  initialize {
    ecrobot_set_light_sensor_activ
    event linefollower:initialized
  }
```

```
doc This is the cyclic task that is called every 1ms to do the actual contro
task run cyclic prio = 2 every = 2 {
  stateswitch linefollower
    state running
      int32 light = 0;
      light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
    default
      <noop>;
}

doc This procedure actually configures the motors based on the speed values
void updateMotorSettings( int left, int right ) {
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, left, 1);
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
}
```

SOLUTION

```
exported interface MotorControl {
  void stop(   );
  void setLeftSpeed( int8 speed );
  void setRightSpeed( int8 speed );
}

exported component Motors {
  provides motorControl : MotorControl;
}
```

```
exported component implementation MotorsNXT : Motors {

  procedure void motorControl.stop( ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, 0, 1);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, 0, 1);
  }

  procedure void motorControl.setLeftSpeed( int8 speed ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, speed, 1);
  }

  procedure void motorControl.setRightSpeed( int8 speed ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, speed, 1);
  }
}
```

SOLUTION

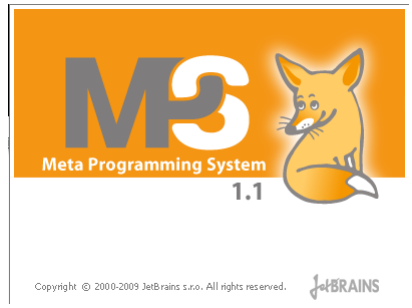
```
module impl imports <<imports>> {

  int speed( int val ) {
    return 2 * val;
  }

  robot script stopAndGo
    block main on bump block retreat on bump <no bumpReaction>
                          stop
                          accelerate to 0 - 30  within 2000
                          drive on for 2000
                          decelerate to 0  within 1000
                          stop
      accelerate to speed(25) within 3000
      drive on for 2000
      turn left for 2000
      block driveMore on bump <no bumpReaction>
        accelerate to 80  within 2000
        turn right for 3000
      decelerate to 0  within 3000
      stop

}
```

TOOLS



JetBrains MPS